[5.5pts] **Question 1) Asymptotic Analysis.**

A) [1.5pts] List *two* different methods we can use to *prove* an asymptotic relation. For each of the two methods, construct a formal mathematical expression as defined per that method to prove/disprove $3^n = O(2^n)$. No need to solve, just construct a formal expression which we can solve to prove or disprove the asymptotic relation.

Method	**Formal Notation - using $3^n = O(2^n)$.**
1)	
2)	

B) [3 pts] Given the following table which includes different asymptotic classes in a random order:

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**
$O(n^2)$	$O(n^3 logn)$	$O(nlogn)$	$O(n)$	$O(n^2 logn)$	$O(n^5)$	$O(2^n)$	$O(n^3)$	$O(logn)$	$O(1)$	$O(n^4)$

Match each the following running time t(n)s with the **tightest possible upper bound**. When answering, please use the alphabets provided for each asymptotic class (not the actual running time) as shown in the sample bullet; also note that you may refer to any asymptotic class more than once.

	function	**Tightest upper-bound**
0	$t(n) = 1000$	J
1	$t(n) = nlog_2(n^2) + n$	
2	$t(n) = 20000^2 + nlogn + n$	
3	$t(n) = n^{3/2} + n^{3/2}$	
4	$t(n) = 100n + nlogn + n^2/2$	
5	$t(n) = n(\log(n^4) - logn) + n^3$	
6	$t(n) = n^2 * log_2\left(\frac{n}{2}\right) + n^2$	
7	$t(n) = 1/2(n^{1/3}) + logn$	
8	$t(n) = n * (100n + 200n^3) + n^3$	
9	$t(n) = (n^3 + n^1)/n$	
10	$t(n) = nlog_4(n^4) + 3n^2$	
11	$t(n) = log_2(2^n) + n^5$	
12	$t(n) = 256log_{256}n$	

C) [1pt] Answer each of the following statements with either **True / False**:

if $f(n) = n^{2.1}$ and $g(n) = n^2 log_2$n, then $f(n) = O(g(n))$.	
if $f(n) = n^{1/2} + log_2 n$ and $g(n) = log_2$n, then $f(n) = \Theta(g(n))$.	
If $T(n) = 10T\left(\frac{n}{3}\right) + n^3, T(1) = 1$, then $T(n) = O(n^3)$.	
If $T(n) = T(n-1) + O(n), T(1) = 1$, then $T(n) = \Theta(n^2)$.	

Question 2) Divide-and-Conquer and Recurrences.

A) Given the following general form of a divide-and-conquer recurrence:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

[1.5pts] Fill out the following table:

Question	Answer
What does T(n) represent?	Running time of the recursive algorithm.
What does the value of a represent?	
What does the value of b represent?	
What does f(n) represent?	

B) [1.5pts] For each of the following code snippets, write the recurrence relation which represents the code (you don't need to solve the recurrence):

Code	Recurrence Relation

`int func1(Arr) {`
`  n = Arr.length;`
`  if (n == 1) then return Arr[0];`
`  int mid = (int) n/2;`
`  int [] Arr1 = new int[mid];`
`  int [] Arr2 = new int[n-mid];`
`  for (int i=0; i<= mid-1; i++)`
`  for (int j=0; j<=1; j++) {`
`  if (j==0) Arr1[i]=Arr2[2i];`
`  else Arr2[i]= Arr[2i+1];`
`  }`
`  Arr2[n-mid-1] = Arr[n-1];`
 	
`  int val1 = func1(Arr1);`
`  int val2 = func1(Arr2);`
 	
`  if (val1 > val2) return val1;`
`  else return val2;`
`}`	

Next page	

```
float func2(Arr) {
   n = Arr.length;
   if (n == 1) then return Arr[0];
   let Arr1, Arr2, Arr3, Arr4 be arrays of size n/2
   for (int i=0; i<= (n/2)-1; i++)
      for (int j=0; j <= (n/2)-1; j++) {
         Arr1[i] = Arr[i];
         Arr2[i] = Arr[i+j];
         Arr3[i] = Arr[n/2+j];
         Arr4[i] = Arr[j];
        }

   int val1 = func1(Arr1);
   int val2 = func1(Arr2);
   int val3 = func1(Arr3);
   int val4 = func1(Arr4);

   return val1*val2*val3*val4;
  }
```

C) [1.5pt] Complete the following divide and conquer algorithm to determine if a set of *n* integers are equal. The initial call would be **`equalSet(S, 0, S.length-1)`** , where **`S`** is an all integers array (there is an easy iterative algorithm to do this checking which would iterate through array elements and compare, but we are solving the problem using a divide-and-conquer approach).

```
boolean equalSet(int [] Arr, int l, int r) {
     if l >= r then
          return true
     else if Arr[l]!= Arr[r] then
          return false;

     int mid = ____1____;              // getting ready to divide,
     return ____2____ && ____3____; // fill in the 3 blanks
    }
```

Blank 1	
Blank 2	
Blank 3	

[4pts] **Question 3) Algorithm Design Techniques.**

A) [1.25pts] Suppose you are given a set T = $\{t_1, t_2. \ldots t_n\}$ of tasks, where a **t**ask t_i requires p_i units of **p**rocessing time to complete. You can run only one task at a time. Let c_i be the **c**ompletion time of task t_i, that is, the time at which task t_i completes processing.

Your goal is to minimize *average* completion time, that is:

$$minimize\ (1/n)\sum_{i=1}^{n} c_i.$$

For example, suppose there are two tasks t_1 with $p_1 = 3$ and t_2 with $p_2 = 5$ and consider the schedule in which t_1 runs first, then $c_1=3$, $c_2 = 8$, and the average completion time is (3+8)/2=5.5. If t_2 runs first, then $c_2 = 5$ and $c_1 = 8$, and the average completion time is (5 + 8)/2 = 6.5.

i) [.25pts] Construct an optimal schedule for the following tasks and their processing times:

T	1	2	3	4	5	6
P	3	8	6	2	5	4

ANSWER:

ii) [.5pts] Describe a *greedy* algorithm that would schedule the tasks so as to minimize the average completion time.

ANSWER:

iii) [.25pts] What would be the running time of your greedy algorithm in part (b)? why?

ANSWER:

iv) [.25pt] Can we solve this problem using a DP approach? (yes/no)

ANSWER:

A) [1pt] Given the following recursive definition:

```
V[0] = 1

V[n] = V[n-1] + V[floor(n/2)] + 1
```

We can write an algorithm to compute the first n values in this series, examples:

V[2] = V[1] + V[1] + 1 = 3,
V[3] = V[2] + V[1] + 1 = 5, etc.

The most straightforward way would be to write a recursive algorithm based directly on the equation definition. Answer the following two question:

i) Writing code based directly on the equation definition will result in an inefficient algorithm, why? What is the problem?

ANSWER:

ii) How can we improve the algorithm?

ANSWER:

C) [.5pt] Using the DP algorithm we have studied in class, finish the last row in the following LCS problem (shaded row):

		L	O	N	G	E	S	T
	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	1	1
T	0	0	0	0	0	0	1	2
O	0	0	1	1	1	1	1	2
N	0	0	1	2	2	2	2	2
E								

D) [.5pt] **i)** Using the DP algorithm we have studied in class, fill in the missing value for the following knapsack problem:

Capacity j →			**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
item	**value**	**weight**	0	0	0	0	0	0	0	0	0	0	0
1	**10**	**5**	0	0	0	0	0	10	10	10	10	10	10
2	**40**	**4**	0	0	0	0	40	40	40	40	40		
3	**30**	**6**	0	0	0	0	40	40	40	40	40	50	50
4	**50**	**3**	0	0	0						90	90	90

ii) [.5pts] Given the above knapsack instance, the maximum value we can achieve is 90. Which items do we include in the knapsack in this case?

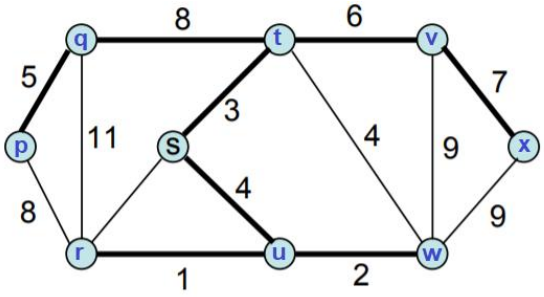
ANSWER:

iii) [.25pts] What is the size of the knapsack in the above instance?

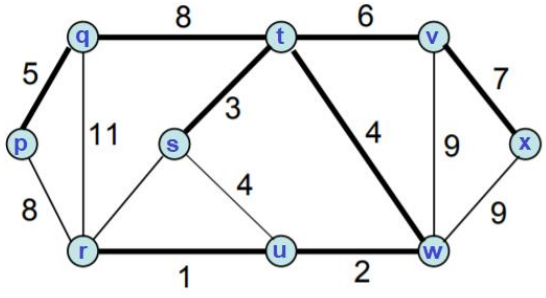
ANSWER:

[2.5pts] **Question 5) Graph Algorithms.**

A) In this question, I am showing the result of applying Prim's algorithm and Kruskal's algorithm, on the same graph, and the bold edges are showing the resultant MST of each algorithm.



Prim's MST algorithm
(starting at s)



Kruskal's MST algorithm

i) [1pt] I am interested in knowing the order by which we are adding edges to the tree. Trace the two algorithms on your own, then fill out the following tables with the correct order by which you are adding edges to the MST:

Prim's:

First edge added	(s,t)
2nd	
3rd	
4th	
5th	
6th	
7th	
8th	(p,q)

Kruskal's:

First edge added	(r,u)
2nd	
3rd	
4th	
5th	
6th	
7th	
8th	(q,t)

ii) [.25pt] Why is the number of edges is the same, in both resultant MSTs?

ANSWER:

iii) [1.25pt] What is the main difference between Dikstra's algorithm and Floyed Warshall's algorithm in terms of the following:

	Dikstra's	Floyed Warshal's
The problem which the algorithm solves		
The design method of the algorithm		
Complexity (worst-case running time)		

[3.5 pts] **Question 6) NPComplexity and Solving NP-Hard Problems.**

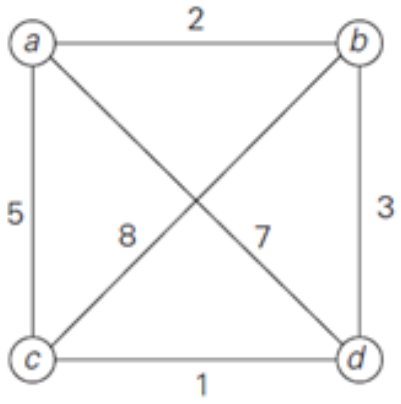
A) [2 pts] Answer each of the following statements with either **True / False**:

Algorithms are fun!	**True**
Every computational problem on input size n can be solved by an algorithm with polynomial running time.	
P and NP are two different complexity classes, but we know that they are equal.	
If we can solve any one of the NPComplete problems by a polynomial time, then all computational problems can be solved by a polynomial time algorithm.	
NPCompletness does not apply to optimization problems.	
Applying brute-force method to combinatorial (hard) problems will result in an exhaustive search algorithm which can never be practical since it will be exponential in running time.	
Applying either backtracking or branch-and-bound methods to combinatorial (hard) problems will always result in polynomial time algorithm.	
Branch-and-bound is applicable only to optimization problems, while backtracking is applicable to non-optimization problems.	
Optimization problems are simpler than decision problems.	

B) [.75pt] Apply backtracking method to solve the following instance of the sub-set sum problem, A = {2, 4, 6} and d = 6. Show me the final state-space tree, make sure you mark any **terminated paths** and mark **all** solutions.

ANSWER:

C) [.75pt] In this question, we are going to solve the following instance of traveling salesman problem using the same branch-and-bound technique we studied in class. The following graph is the input, where node *a* represents the starting city:



Again, we will be using the same formula for computing the lower bound function as used lecture: $lb = \lceil s/2 \rceil$, where s in equal to the sum of nearest distances in which each node adds up the distance to the nearest two cities. We will also be applying the same constraint: consider tours in which node b appears before node c, to terminate duplicate tours which differ only by direction.

Answer the following questions which refer to the next figure:

i) What should be the value of the *lb* function at node **(3)**?

ANSWER:

ii) What should be the value of the tour length *l* function at node **(5)**?

ANSWER:

iii) Why is the path at node **(2)** terminated?

ANSWER: